```
/**
 * Title:        Finger Print Identification
 * Description:  Third Version for fingerprint version on smart card. 1.5 MINUTES VERSION
 * The matching engine on the smart card.
 * Compiled size: approx. 5K bytes.
 * Copyright:    Copyright (c) 2001
 * Company:      CSP,NTU
 * @author Lawrence CHEN
 * @version 1.0
 */
/*
 * Package name
 */
package com.gemplus.examples.oppurseV1;

/*reset

 * Imported packages
 */
import javacard.framework.*;
import visa.openplatform.*;


public class OPPurseV1 extends javacard.framework.Applet
{
   // the APDU constants for all the commands.
          // The following are the APDU commands in byte code form.
          private final static byte INS_GET_BALANCE          = (byte)0x30 ;
          private final static byte INS_DEBIT                = (byte)0x31 ;
          private final static byte INS_CREDIT               = (byte)0x32 ;
          private final static byte INS_VERIFY_PIN           = (byte)0x33 ;
          private final static byte SETCOUNTER               = (byte)0x34 ;

          // Registration related command. 0x3X
          private final static byte SetMode                          = (byte)0x35 ;
          private final static byte ReceiveMainTemplate              = (byte)0x36 ;
          /* The ReceiveMainTemplate(...) function is used to receive the secured portion of the
          minutiae template from the PC terminal during enrolment. The received template will be stored
          in the secured memory and will never be released to the outside of the smartcard. During
          verification, this template will be transformed to aligned enrolment template first, and then the
          aligned template will be used to compare against the aligned query template to find out the
          similarity (matching score) inside the smartcard. All operations related to this secured portion
          of minutiae template shall only be executed inside the smartcard. In this source code, the
          secured portion of the minutiae template is named as main_template.
          */

          private final static byte SaveBaseTemplate         = (byte)0x37 ;
          /* The SaveBaseTemplate(...) function is used to receive the open portion of the minutiae
          template from the PC terminal during enrolment. This template can be encoded in either
          compressed or non-compressed format. The open template contains relative information
          among minutiae which can be used to find out the alignment information between the
          enrolment template and the query template in the PC terminal. During verification, this
          template is sent out to the PC terminal. The PC terminal can use this open portion to compare
          the relative information of the query template to find out the alignment information such as
          offset and orientation difference. The PC terminal can use the alignment information to
          perform affine transform or polar transform to align both the enrolment template and the
          query template with the same reference coordinate for direct minutiae template comparison.
          As this open portion of minutiae template contains only relative information among minutiae,
          it is difficult to use this open portion to reverse engineer the coordinate of the secured portion
```

```
        of the minutiae template. In this source code, the open portion of minutiae template is named
        as base_template or open template.
        */

        private final static byte SetUserInfoC          = (byte)0x38 ;
        private final static byte SavePhoto             = (byte)0x39 ;
        private final static byte LockUserInfo                  = (byte)0x3A ;
//      private final static byte Test_Buffer                   = (byte)0x3B ;
//      private final static byte matchm                        = (byte)0x3C ;

        // Authentication related command. 0x4X
        private final static byte GetBaseTemplateInfoC    = (byte)0x41 ;
        /* The GetBaseTemplateInfoC (...) function is used to get the size of the open template, the
        number of blocks which need to be received in PC and the size of each block.
        */

        private final static byte SendBaseTemplate                  = (byte)0x42 ;
        /* The SendBaseTemplate (...) function is used to send the open template from the smartcard
        to the PC terminal for alignment.
        */

        private final static byte SendPartialMainC        = (byte)0x43 ;
        /*  This function SendPartialMain(...) is used to get certain information of minutiae as auxiliary
        information for alignment. This function is an optional function which is not used in later
        version. */

        private final static byte GetAuxilaryMinutiaeC    = (byte)0x44 ;
        /*This function is used to get the index of particular minutia in enrolment template which has
        been identified in the alignment process in the PC terminal. This function records down the
        index and find the corresponding coordinate of minutia in the enrolment template (inside the
        secured portion of minutiae template) for later on-card alignment process during matching.
        This function will not release any information related to secured template to the outside world.
        */

        private final static byte LoadMatchingVariablesC  = (byte)0x45 ;
        /* The LoadMatchingVariablesC (...) function is used to load the aligned query template from
        the PC terminal after the alignment process is completed on the PC side. The template will be
        stored in the volatile memory in the smartcard. This template will be used to calculate the
        matching score during the matching process.
        */

        private final static byte Match                             = (byte)0x46 ;
        /* The Match (...) function is used to match the aligned query template which has be received
        by using LoadMatchingVariables(...) function. Before the computation of matching, the on-
        card alignment procedure for the secured portion of enrolment template will be executed
        inside the smartcard first. Once the alignment of the enrolment template is completed, the
        smartcard will collate the aligned enrolment template with the aligned query template in order
        to find out the overall similarity as matching score. The matching score will then be compared
        to the internal security threshold to determine the query is from genuine user or imposter to
        grant or deny the subsequent transaction respectively.
        */

        private final static byte GetMatchScoreC                = (byte)0x47 ;
        private final static byte AuthenticateC                 = (byte)0x48 ;
        private final static byte GetUserInfoC                  = (byte)0x49 ;
        private final static byte GetPhotoC                             = (byte)0x4A ;
        private final static byte UnlockUserInfo        = (byte)0x4B ; // Must delect for release version

    // the OP/VOP specific instruction set for mutual authentication
        private final static byte CLA_INIT_UPDATE       = (byte)0x80 ;
```

```
        private final static byte INS_INIT_UPDATE          = (byte)0x50 ;
        private final static byte CLA_EXTERNAL_AUTHENTICATE = (byte)0x84 ;
        private final static byte INS_EXTERNAL_AUTHENTICATE = (byte)0x82 ;


        private final static short atan_value[] =
        {
        0,57,115,172,229,286,343,401,457,514,571,628,684,741,797,853,
        909,965,1020,1076,1131,1186,1241,1295,1350,1404,1458,1511,1564,1617,1670,1722,
        1775,1826,1878,1929,1980,2031,2081,2131,2180,2229,2278,2327,2375,2423,2470,2517,
        2564,2611,2657,2702,2748,2792,2837,2881,2925,2968,3011,3054,3096,3138,3180,3221,
        3262,3302,3343,3382,3422,3461,3499,3538,3575,3613,3650,3687,3724,3760,3796,3831,
        3866,3901,3935,3969,4003,4037,4070,4102,4135,4167,4199,4230,4262,4292,4323,4353,
        4383,4413,4442,4471,4500
        };
        private final static short sin_value[] =
        {
        0,18,35,53,70,88,105,122,140,157,174,191,
        208,225,242,259,276,293,310,326,343,359,375,391,407,423,439,454,470,
        485,500,516,530,545,560,574,588,602,616,630,643,657,670,682,695,707
        };

// the PIN validity flag
private boolean validPIN = false;

// SW bytes for PIN Failed condition
        // the last nibble is replaced with the number of remaining tries
        private final static short  SW_PIN_FAILED = (short)0x63C0;

// the illegal amount value for the exceptions.
private final static short ILLEGAL_AMOUNT = 1;

// the maximum balance in this purse.
private static final short maximumBalance = 10000;

// the current balance in this purse.
private short balance;

        //counter
        private short counter = (short)(0x0030);

/* Security part of declarations */

// the Security Object necessary to credit the purse
private ProviderSecurityDomain securityObject = null;

// the security channel number
byte secureChannel = (byte)0xFF;

// the authentication status
private boolean authenticationDone = false;

// the secure channel status
private boolean channelOpened = false;

        private byte CardEnabled = 0;


/**
 * Only this class's install method should create the applet object.
```

```
 */
protected OPPurseV1(byte[] buffer, short offset, byte length)
{
   // data offset is used for application specific parameter.
   // initialization with default offset (AID offset).
   short dataOffset = offset;

   if(length > 9) {
      // Install parameter detail. Compliant with OP 2.0.1.

      // | size | content
      // |------|-------------------------
      // |  1   | [AID_Length]
      // | 5-16 | [AID_Bytes]
      // |  1   | [Privilege_Length]
      // | 1-n  | [Privilege_Bytes] (normally 1Byte)
      // |  1   | [Application_Proprietary_Length]
      // | 0-m  | [Application_Proprietary_Bytes]

      // shift to privilege offset
      dataOffset += (short)( 1 + buffer[offset]);
      // finally shift to Application specific offset
      dataOffset += (short)( 1 + buffer[dataOffset]);
      // checks wrong data length
      if(buffer[dataOffset] != 2)
         // return received proprietary data length in the reason
         ISOException.throwIt((short)(ISO7816.SW_WRONG_LENGTH + offset + length - dataOffset));

      // go to proprietary data
      dataOffset++;

   } else {
      // Install parameter compliant with OP 2.0.
      if(length != 2)
         ISOException.throwIt((short)(ISO7816.SW_WRONG_LENGTH + length));
   }
               // retreive the balance value from the APDU buffer
   short value = (short)(((buffer[(short)(dataOffset + 1)]) & 0xFF)
                        | ((buffer[dataOffset] & 0xFF) << 8));

   // checks initial balance value
   if(value > maximumBalance)
      ISOException.throwIt(ISO7816.SW_DATA_INVALID);

               // initializes the balance with the APDU buffer contents
   balance = value;

   // register this instance as an installed Applet
   register();

   // ask the system for the Security Object associated to the Applet
   securityObject = OPSystem.getSecurityDomain();

   // applet is personalized and its state can change
   OPSystem.setCardContentState(OPSystem.APPLET_PERSONALIZED);

   // build the new ATR historical bytes
   byte[] newATRHistory = new byte[]
   {
      // put "OPPurse" in historical bytes.
```

```
        (byte)0x4F, (byte)0x50, (byte)0x50, (byte)0x75, (byte)0x72, (byte)0x73, (byte)0x65
    };
    // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    // !!! ACTIVATED IF INSTALL PRIVILEGE IS "Default Selected" (0x04). !!!
    // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    // change the default ATR to a personalized's one
    OPSystem.setATRHistBytes(newATRHistory, (short)0, (byte)newATRHistory.length);
}

/**
 * Method installing the applet.
 * @param installparam the array constaining installation parameters
 * @param offset the starting offset in installparam
 * @param length the length in bytes of the data parameter in installparam
 */
public static void install(byte[] installparam, short offset, byte length )
throws ISOException
{
    // applet  instance creation with the initial balance
                    // The only method can be called by install method if the class constructor.
                    // The constructor should be either private or protected.
    new OPPurseV1(installparam, offset, length );
}

/**
 * Select method returning true if applet selection is supported.
 * @return boolean status of selection.
 */
public boolean select()
{
    validPIN = false;
                    //ValidUser = 0;
                    CardEnabled = 0;
    // reset security if used.
    // In case of reset deselect is not called
    reset_security();
    // return status of selection
    return true;
}

/**
 * Deselect method.
 */
public void deselect()
{
    // reset security if used.
    reset_security();
    return;
}

/**
 * Method processing an incoming APDU.
 * @see APDU
 * @param apdu the incoming APDU
 * @exception ISOException with the response bytes defined by ISO 7816-4
 */
public void process(APDU apdu) throws ISOException
{
    // get the APDU buffer
    // the APDU data is available in 'apduBuffer'
```

```
        byte[] apduBuffer = apdu.getBuffer();

        // the "try" is mandatory because the debit method
        // can throw a javacard.framework.UserException
        try
        {
                if(apduBuffer[ISO7816.OFFSET_INS]>(byte)0x34 ||
apduBuffer[ISO7816.OFFSET_INS]<(byte)0x4F)
                                {
                                        switch(apduBuffer[ISO7816.OFFSET_INS])
                                        {
                                        //****** Registration related commands.
                                        case SetMode:
                                                SetCurrentMode(apdu);
                                                break;
                                        case ReceiveMainTemplate:
                                                receive_main_template(apdu);
                                                break;
                                        case SaveBaseTemplate:
                                                save_base_template(apdu);
                                                break;
                                        case SetUserInfoC:
                                                SetUserInfo(apdu);
                                                break;
                                        case SavePhoto:
                                                save_photo(apdu);
                                                break;
                                        case LockUserInfo:
                                                lockUserInfo(apdu);
                                                break;
                                        //****** Matching Related commands.
                                        case GetBaseTemplateInfoC:
                                                GetBaseTemplateInfo(apdu);
                                                break;
                                        case SendBaseTemplate:
                                                send_base_template(apdu);
                                                break;
                                        case SendPartialMainC:
                                                SendPartialMain(apdu);
                                                break;
                                        case GetAuxilaryMinutiaeC:
                                                GetAuxilaryMinutiae(apdu);
                                                break;
                                        case Match:
                                                match(apdu);
                                                break;
                                        case GetMatchScoreC:
                                                GetMatchScore(apdu);
                                                break;
        //                              case AuthenticateC:
        //                                      authenticate(apdu);
        //                                      break;
                                        case GetUserInfoC:
                                                GetUserInfo(apdu);
                                                break;
        //                              case GetPhotoC:
        //                                      GetPhoto(apdu);
        //                                      break;
                                        // Must Delete UnlockUserInfo entry point for the release version.
                                        case UnlockUserInfo :
```

```
                                        UnlockUserInfo(apdu);
                                        break;
default :
   // The INS code is not supported by the dispatcher
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED) ;
break ;

                                }// end of switch.
                        }// endif.
                        else
                        {
                        switch(apduBuffer[ISO7816.OFFSET_INS])
{
case INS_VERIFY_PIN :
    verifyPIN(apdu);
break ;

case INS_GET_BALANCE :
    getBalance(apdu) ;
break ;

case INS_DEBIT :
    debit(apdu) ;
break ;

case INS_CREDIT :
    credit(apdu) ;
break ;

                                case SETCOUNTER :
                                        SetCounter(apdu);

                                break;

case INS_INIT_UPDATE :
   if(apduBuffer[ISO7816.OFFSET_CLA] == CLA_INIT_UPDATE)
      // call initialize/update security method
       init_update(apdu) ;
   else
      // wrong CLA received
      ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
break ;

case INS_EXTERNAL_AUTHENTICATE :
   if(apduBuffer[ISO7816.OFFSET_CLA] == CLA_EXTERNAL_AUTHENTICATE)
      // call external/authenticate security method
    external_authenticate(apdu) ;
   else
      // wrong CLA received
      ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
break ;

case ISO7816.INS_SELECT :
break ;

default :
   // The INS code is not supported by the dispatcher
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED) ;
break ;
}       // end of the switch
```

```
                        } //end of if.
    } // end of the try
                catch(UserException e)
    {
       // translates the UserException in an ISOException.
                    if(e.getReason() == ILLEGAL_AMOUNT)
          throw new ISOException ( ISO7816.SW_DATA_INVALID ) ;
                }
}

//---------------------------------------------------------------------
//-            P R I V A T E    M E T H O D S                 -
//---------------------------------------------------------------------

        /**
         * Handles Verify Pin APDU.
         *
         * @param apdu APDU object
         */
        private void verifyPIN(APDU apdu)
        {
    // get APDU data
                apdu.setIncomingAndReceive();
    // get APDU buffer
    byte[] apduBuffer = apdu.getBuffer();
    // check that the PIN is not blocked
    if(OPSystem.getTriesRemaining() == 0)
       OPSystem.setCardContentState(OPSystem.APPLET_BLOCKED);

    // Pin format for OP specification
    //
    // |type(2),length|nible(1),nible(2)|nible(3),nible(4)|...|nible(n-1),nible(n)|
    //
    // get Pin length
    byte length = (byte)(apduBuffer[ISO7816.OFFSET_LC] & 0x0F);
    // pad the PIN ASCII value
    for(byte i=length; i<0x0E; i++)
    {
       // only low nibble of padding is used
       apduBuffer[ISO7816.OFFSET_CDATA + i] = 0x3F;
    }
    // fill header TAG
    apduBuffer[0] = (byte)((0x02 << 4) | length);
    // parse ASCII Pin code
    for(byte i=0; i<0x0E; i++)
    {
       // fill bytes with ASCII Pin nibbles
       if((i & 0x01) == 0)
          // high nibble
          apduBuffer[(i >> 1)+1] = (byte)((apduBuffer[ISO7816.OFFSET_CDATA + i] & 0x0F) << 4);
       else
          // low nibble
          apduBuffer[(i >> 1)+1] |= (byte)(apduBuffer[ISO7816.OFFSET_CDATA + i] & 0x0F);
    }
    // verify the received PIN
    // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    // !!! WARNING PIN HAS TO BE INITIALIZED BEFORE USE !!!
    // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    if(OPSystem.verifyPin(apdu, (byte)0))
    {
```

```
        // set PIN validity flag
        validPIN = true;
        // if applet state is BLOCKED then restore previous state (PERSONALIZED)
        if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
            OPSystem.setCardContentState(OPSystem.APPLET_PERSONALIZED);
        return;
                }
         // the last nibble of returned code is the number of remaining tries
                ISOException.throwIt((short)(SW_PIN_FAILED + OPSystem.getTriesRemaining()));
        }

/**
 * Performs the "getBalance" operation on this counter.
 *
 * @param apdu The APDU to process.
 */
private void getBalance( APDU apdu )
{
    // check valid Applet state
    if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

    // get the APDU buffer
    byte[] apduBuffer = apdu.getBuffer();

    // writes the balance into the APDU buffer after the APDU command part
                apduBuffer[5] = (byte)(balance >> 8) ;
                apduBuffer[6] = (byte) balance;

    // sends the APDU response
    // switches to output mode
                apdu.setOutgoing() ;
    // 2 bytes to return
                apdu.setOutgoingLength((short)2) ;
    // offset and length of bytes to return in the APDU buffer
                apdu.sendBytes((short)5, (short)2) ;
}

        private void SetCounter(APDU apdu)
        {
    if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                byte[] apduBuffer = apdu.getBuffer();
    // writes the counter into the APDU buffer after the APDU command part
                apduBuffer[5] = (byte)(counter >> 8);
                apduBuffer[6] = (byte)(counter);
                counter ++;
    // sends the APDU response
    // switches to output mode
                apdu.setOutgoing() ;
    // 2 bytes to return
                apdu.setOutgoingLength((short)2) ;
    // offset and length of bytes to return in the APDU buffer
                apdu.sendBytes((short)5, (short)2) ;
        }

/**
 * Performs t "debit" operation on this counter.
 *
 * @param apdu The APDU to process.
```

```
 * @exception ISOException If the APDU is invalid.
 * @exception UserException If the amount to debit is invalid.
 */
private void debit(APDU apdu) throws ISOException, UserException
{
   // check valid Applet state
   if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                              ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

   // the operation is allowed only if master pin is validated
         if(!validPIN)
      ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

   // get the APDU buffer
   byte[] apduBuffer = apdu.getBuffer();

         // Gets the length of bytes to recieved from the terminal and receives them
   // If does not receive 4 bytes throws an ISO.SW_WRONG_LENGTH exception
                  if(apduBuffer[4] != 2 || apdu.setIncomingAndReceive() != 2)
   {
                     ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
   }
                  // Reads the debit amount from the APDU buffer
   // Starts at offset 5 in the APDU buffer since the 5 first bytes
   // are used by the APDU command part
                  short amount = (short)(((apduBuffer[6]) & (short)0x000000FF)
      | ((apduBuffer[5] << 8 ) & (short)0x0000FF00));

   // tests if the debit is valid
   if((balance >= amount) && (amount > 0))
   {
      // does the debit operation
      balance -= amount ;

      // writes the new balance into the APDU buffer
      // (writes after the debit amount in the APDU buffer)
      apduBuffer[7] = (byte)(balance >> 8) ;
      apduBuffer[8] = (byte)balance ;

      // sends the APDU response
      apdu.setOutgoing() ; // Switches to output mode
      apdu.setOutgoingLength((short)2) ; // 2 bytes to return
      // offset and length of bytes to return in the APDU buffer
      apdu.sendBytes((short)7, (short)2) ;
   }
   else
      // throw a UserException with illegal amount as reason
      throw new UserException(ILLEGAL_AMOUNT) ;
}

/**
 * Performs the "credit" operation on this counter. The operation is allowed only
 * if master pin is validated
 *
 * @param apdu            The APDU to process.
 * @exception ISOException If the APDU is invalid or if the amount to credit
 *                      is invalid.
 */
private void credit(APDU apdu) throws ISOException
{
```

```
    // check valid Applet state
    if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                            ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

    // the operation is allowed only if master pin is validated and authentication is done
            if (!validPIN || !authenticationDone)
       ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

    // get the APDU buffer
    byte[] apduBuffer = apdu.getBuffer();

                    // gets the length of bytes to recieved from the terminal and receives them
    // if does not receive 2 bytes throws an ISO.SW_WRONG_LENGTH exception
                    if(apduBuffer[4] != 2 || apdu.setIncomingAndReceive() != 2)
       throw new ISOException(ISO7816.SW_WRONG_LENGTH) ;

                    // reads the credit amount from the APDU buffer
    // starts at offset 5 in the APDU buffer since the 5 first bytes
    // are used by the APDU command part
                    short amount = (short)(((apduBuffer[6]) & (short)0x000000FF)
       | ((apduBuffer[5] << 8) & (short)0x0000FF00));

    // tests if the credit is valid
    if(((short)(balance + amount) > maximumBalance) || (amount <= (short)0))
       throw new ISOException(ISO7816.SW_DATA_INVALID) ;
    else
       // does the credit operation
       balance += amount ;
}

/**
 * Performs the "init_update" security operation.
 *
 * @param apdu The APDU to process.
 */
private void init_update( APDU apdu )
{
    // receives data
    apdu.setIncomingAndReceive();
    // checks for existing active secure channel
    if(channelOpened)
    {
       // close the openned security channel
       try
       {
          securityObject.closeSecureChannel(secureChannel);
       }
       catch(CardRuntimeException cre2)
       {
          // channel number is invalid. this case is ignored
       }
       // set the channel flag to close
       channelOpened = false;
    }
    try
    {
       // open a new security channel
       secureChannel = securityObject.openSecureChannel(apdu);
       // set the channel flag to open
       channelOpened = true;
```

```
            // get expected length
            short expected = apdu.setOutgoing();
            // send authentication result
            // expected length forced to 0x1C
            apdu.setOutgoingLength((byte)0x1C);
            apdu.sendBytes(ISO7816.OFFSET_CDATA, (byte)0x1c);
        }
        catch(CardRuntimeException cre)
        {
            // no available channel or APDU is invalid
            ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
        }
    }
/*
        private void GetUserInfo(APDU apdu)
        {
                byte[] buffer = apdu.getBuffer();
                byte length,i;
        if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                if(ValidUser!=0x34)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                else
                {
                        buffer[4] = (byte)(name.length+UserID.length);
                        length = (byte)(name.length);
                        for(i=0;i<length;i++)
                                buffer[5+i]=name[i];
                        length = (byte)(5+length);
                        for(i=0;i<UserID.length;i++)
                                buffer[(byte)(length+i)] = UserID[i];
                //apdu.setOutgoing() ;
        // 2 bytes to return
                //apdu.setOutgoingLength((short)30) ;
        // offset and length of bytes to return in the APDU buffer
                //apdu.sendBytes((short)5, (short)30) ;
                apdu.setOutgoingAndSend((short)5, (short)30) ;
                }

        }
*/
        /*
        !!!!!!!NOTE!!!!!!
        This function is for testing purpose only and SHOULD be deleted after finish development.
        */

        private void GetMatchScore( APDU apdu )
    {
        // check valid Applet state
        //if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                //      ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

        // get the APDU buffer
        byte[] apduBuffer = apdu.getBuffer();

        // writes the balance into the APDU buffer after the APDU command part
                apduBuffer[5] = (byte)MatchScore ;
                apduBuffer[6] = (byte)0x7A;
        // sends the APDU response
        // switches to output mode
```

```
                //apdu.setOutgoing() ;
    // 2 bytes to return
                //apdu.setOutgoingLength((short)2) ;
    // offset and length of bytes to return in the APDU buffer
                //apdu.sendBytes((short)5, (short)2) ;
                apdu.setOutgoingAndSend((short)5,(short)2);
}

/**
 * Performs the "external_authenticate" security operation.
 *
 * @param apdu The APDU to process.
 */
private void external_authenticate( APDU apdu )
{
    // receives data
    apdu.setIncomingAndReceive();
    // checks for existing active secure channel
    if(channelOpened)
    {
        try
        {
            // try to authenticate the client
            securityObject.verifyExternalAuthenticate(secureChannel, apdu);
            // authentication succeed
            authenticationDone = true;
        }
        catch(CardRuntimeException cre)
        {
            // authentication fails
            // set authentication flag to fails
            authenticationDone = false;
            // close the openned security channel
            try {
                securityObject.closeSecureChannel(secureChannel);
            } catch(CardRuntimeException cre2) {
                // channel number is invalid. this case is ignored
            }
            // set the channel flag to close
            channelOpened = false;
            // send authentication result
            ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
        }
        // send authentication result
        ISOException.throwIt(ISO7816.SW_NO_ERROR);
    }
    else
        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
}

/**
 * The "reset_security" method close an opened secure channel if exist.
 * @return void.
 */
private void reset_security()
{
    // close the secure channel if openned.
    if(secureChannel != (byte)0xFF)
    {
        try
```

```
        {
            // close the openned security channel
            securityObject.closeSecureChannel(secureChannel);
        }
        catch(CardRuntimeException cre2)
        {
            // channel number is invalid. this case is ignored
        }
        // reset security parameters
        secureChannel = (byte)0xFF;
        channelOpened = false;
        authenticationDone = false;
    }
                // Reset all matching constants to avoid any previous succesful login
                // continue to next secession.
                        UserValid = 0;
                        UserVerifyDone = 0;
                        MatchScore = 0;
                    FIDmode = -1;
    return;
  }

     private void SetCurrentMode(APDU apdu)
     {
                byte buffer[] = apdu.getBuffer();
                short Mindex;
                short SA1,SA2;
                byte P1,P2,LC,length;
                //short SA1,SA2;
                P1 = buffer[ISO7816.OFFSET_P1];
                P2 = buffer[ISO7816.OFFSET_P2];
                LC = buffer[ISO7816.OFFSET_LC];
//    if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
//                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                //CheckSum = 0;
                apdu.setIncomingAndReceive();
                FIDmode =0; // set to default mode 0;
                //if(LC != 4)
                //      ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if( P1 != (~P2))
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                //if(InitObjDone==0)
                //      ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
                if(P1 == 0x27)
                        FIDmode =1; // registration
                if(P2 == 0x4A)
                        FIDmode =2; // verification.
                //if(FIDmode == 1)
                //      OwerTemplateValid=0;
                 //length = LC;
                 Mindex =0; // reset the current Minutiae index to zero.

                //SA1 = (short)((short)(buffer[7])& (short)0x00FF);
                SA1 = (short)(buffer[7]);
                //SA1 = (short)(((short)(buffer[8])<<(byte)8)+SA1);
                SA2 = (short)((short)(buffer[5])& (short)0x00FF);
                //SA2 = (short)(((short)(buffer[6])<<(byte)8)+SA2);
                buffer[5] = (byte)(0xFF);
                buffer[6] = (byte)(0x70);
                if(FIDmode == 1)
```

```
                {
                        i_RidgeFreq = (byte)SA1;
                        NOMI = (byte)SA2;
                }
                if(FIDmode == 2)
                {
                        j_RidgeFreq = (byte)SA1;
                        NOMJ = (byte)SA2;
                }
                UserVerifyDone = 0;
                MatchScore = (byte)0xFF;
                buffer[5] = (byte)0xFF;
//              if(EEAlocate == 0)
//                      buffer[6] = (byte)0x71;
//              else
//                      buffer[6] = (byte)0x70;
                buffer[6] = (byte)0x70;
                apdu.setOutgoingAndSend((short)5,(short)2);
        }
/*
        private void FIDverify(APDU apdu) throws ISOException
        {
                byte buffer[] = apdu.getBuffer();
                short SA1;
                byte P1,P2,LC;
                P1 = buffer[ISO7816.OFFSET_P1];
                P2 = buffer[ISO7816.OFFSET_P2];
                LC = buffer[ISO7816.OFFSET_LC];
                //short SA1;
                MatchScore=0;
    if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                //if(LC != 2)
                //      ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if( P1 != (~P2))
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if(P2 != 0x4A)
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if(OwerTemplateValid==0)
                        ISOException.throwIt(ISO7816.SW_FILE_INVALID);
                SA1 = (short)(((short)(buffer[(short)5])<<8) + (short)(buffer[(short)5])&(short)0x00FF);
                //Enable Verify CheckSum later.
                //if(T_chksum!=chksum)
                //      ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
                buffer[5] = (byte)(0xFF);
                buffer[6] = (byte)(0x70);
                Init_Match(Mi_length,Mj_ngth,Mi_ridge_freq,Mj_ridge_freq);
                MatchScore = match((short)22,(byte)5);
                if(MatchScore>(byte)22 && MatchScore<(byte)70)
                        ValidUser=(byte)0x34;
                else
                        ValidUser=(byte)0xFF;

                VerifyDone = 1;
                apdu.setOutgoingAndSend((short)5,(short)2);
                //apdu.setOutgoing();
                //apdu.setOutgoingLength((short)2);
                //apdu.sendBytes((short)5,(short)2);
                return;
        }
```

```
*/
        // Function to Validate User Minutiae Template, User Name and User ID.
/*
        private void RegisterTemplate(APDU apdu) throws ISOException
        {
                byte buffer[] = apdu.getBuffer();
                byte i,j;
      short SA1,SA2;
                byte P1,P2,LC;
                P1 = buffer[ISO7816.OFFSET_P1];
                P2 = buffer[ISO7816.OFFSET_P2];
                LC = buffer[ISO7816.OFFSET_LC];
                apdu.setIncomingAndReceive();
                if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                        ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                if(LC < 2)
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if(P1 != (~P2))
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if(P1 != 0x27)
                        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                SA1 = (short)(((short)(buffer[(short)6])<<8) + (short)(buffer[(short)5])&(short)0x00FF);
                for(i=(byte)0;i<buffer[(byte)7];i++)
                        name[i] = buffer[(byte)(i+8)];
                //size = (byte)(buffer[(byte)4]-temp[(byte)2]-(byte)3);
                j = (byte)buffer[7];
                for(i=(byte)0;i<10;i++)
                        UserID[i] = buffer[(byte)(j+i)];
                OwerTemplateValid=1;
                buffer[5] = (byte)(0xFF);
                buffer[6] = (byte)(0x70);
                apdu.setOutgoingAndSend((short)5,(short)2);
        }
*/
/*
        private void GetTestBuffer(APDU apdu)
        {
                  // short i;SA1
                  //short size;SA2
             if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                          ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                        byte buffer[];

                        buffer = apdu.getBuffer();
                        SA2 = (short) buffer.length;
                    buffer[5] = ME1.Compare_template();
                        buffer[6] = (byte)0xCC;
                        buffer[7] = (byte)0xAA;
                        buffer[8] = (byte)SA2;
                        for(SA1=0;SA1<245;SA1++)
                           buffer[(short)(9+SA1)] = TestBuffer[SA1];
                    apdu.setOutgoingAndSend((short)5,(short)0xFF);
        }
*/

        // Save data from apdu to base_template. (Save the open portion of template into smartcard)
        // APDU input and response: please see receive_template(...).
        // IF the template is locked, function throws SW_COMMAND_NOT_ALLOWED exception.
        /* The save_base_template(...) function is used to receive the open portion of the minutiae
        template from the PC terminal during enrolment. This template can be encoded in either
```

compressed or non-compressed format. The open template contains relative information among minutiae which can be used to find out the alignment information between the enrolment template and the query template in the PC terminal. During verification, this template is sent out to the PC terminal. The PC terminal can use this open portion to compare the relative information of the query template to find out the alignment information such as offset and orientation difference. The PC terminal can use the alignment information to perform affine transform or polar transform to align both the enrolment template and the query template with the same reference coordinate for direct minutiae template comparison. As this open portion of minutiae template contains only relative information among minutiae, it is difficult to use this open portion to reverse engineer the coordinate of the secured portion of minutiae template. In this source code, the open portion of minutiae is named as base_template.
*/

```
    private void save_base_template(APDU apdu) throws ISOException
    {
            if(UserInfoLock==1)
                    ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
       receive_template(apdu,base_template);
    }

    // Save User Photo from apdu to UserPhoto[].
    // APDU input and response: please see receive_template(...).
    // IF the template is locked, function throws SW_COMMAND_NOT_ALLOWED exception.
    private void save_photo(APDU apdu) throws ISOException
    {
            if(UserInfoLock==1)
                    ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
       receive_template(apdu,UserPhoto);
    }

    // receive_template(...)
    // Function: receive data from apdu buffer and save data in array[].
    // APDU Values:
    // P1: Total number of blocks.(N blocks)
    // P2: current block number.(start from 1 to N)
    // P1 and P2 should not equal to zero.
    // if the input data size is larger than the size of an array,
    // SW_FILE_FULL exception will be thrown.
    //          CLA INS P1 P2 LA (BYTES)        .....         LE(return bytes)
    //APDU INPUT: CLA INS BA BN(size)     ...base_template[N]..  04
    //APDU OUTPUT: FF 70 BN size
    // size: should not greater than 127 bytes. last byte should be FF for termination.
    //NOTE : method should not be called directly from APDU command.
    private void receive_template(APDU apdu,byte array[]) throws ISOException
    {
                byte databuffer[] = apdu.getBuffer();
                short i,length;//SA1,SA2,SA3
                short Tindex;
                byte P1,P2;
                short SA1;
                P1 = databuffer[ISO7816.OFFSET_P1]; // Total number of block.
                P2 = databuffer[ISO7816.OFFSET_P2]; // current block.
                apdu.waitExtension();
                SA1 = apdu.setIncomingAndReceive();
                //if(SA1 != databuffer[ISO7816.OFFSET_LC])
                        //  ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
          if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
                          ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
                if(databuffer[ISO7816.OFFSET_LC]==0)
```

```
                    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if( P1>12 || P2>12)
                    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                if( P1 < P2)
                    ISOException.throwIt(ISO7816.SW_DATA_INVALID);
                length = databuffer[ISO7816.OFFSET_LC]; // get the size of the blk buffer.
                if(P2==1)
                    Mindex=0; // Clear Mindex for saving the minutiae template.
                Tindex = Mindex; // copy the Minutiae Template index from EEPROM to Temp Index in
RAM.
                for(i=5;i<(short)(length+5);i++)
                {
                    array[(short)(Tindex++)] = databuffer[i];
                    if(Tindex>array.length)
                        ISOException.throwIt(ISO7816.SW_FILE_FULL);
                }
                Mindex = Tindex;
              databuffer[5] = (byte)(0xFF);
              databuffer[6] = (byte)(0x70);
              databuffer[7] = P2;
              databuffer[8] = (byte)length;
              apdu.setOutgoingAndSend((short)5,(short)4);
        }

        // send_base_template(...) (Send the open portion of the template to the terminal(PC) for
alignment purpose)
        /* The SendBaseTemplate (...) function is used to send the open template from the smartcard
        to the PC terminal for alignment.
        */

        //          CLA INS P1 P2 LA (BYTES)        .....       LE(return bytes)
        // APDU INPUT: CLA INS 00 BN00
BlockSize
        // APDU OUTPUT:  (base_template[N] + FF) Total bytes = 128. (127 data + 1 byte end code(FF))
        // Error: if the input block number BN is larger than the actual blocknumber and P1!=0,
        // SW_WRONG_P1P2 exception will be returned
        // Note: BN[0..maxblk].
      public void send_base_template(APDU apdu) throws ISOException
      {
                byte buffer[] = apdu.getBuffer();
                byte size = buffer[5]; // read the return size.
                byte BN;
        short i,offset;
                BN = buffer[ISO7816.OFFSET_P2];
                if(BN > BlockNumber && buffer[ISO7816.OFFSET_P1]!=0)
                    ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
                offset = (short)(BN * 127);
        for(i=0;i<size;i++)
          buffer[(short)(i+5)] = base_template[(short)(offset + i)];
                //buffer[(short)(i+6)]=(byte)0xFF;
                apdu.setOutgoingAndSend((short)5,size);
      }

        //GetBaseTemplateInfo(...) (Let the terminal(PC) to know how many blocks of data (open
template) need to be transmit from the smartcard to terminal)
        /* The GetBaseTemplateInfoC (...) function is used to get the size of the open template, the
        number of blocks which need to be received in PC and the size of each block.
        */

        //          CLA INS P1 P2 LA (BYTES)        .....       LE(return bytes)
```

```
        //APDU INPUT: CLA INS 00 7C 00          ...base_template[N]..  04
        //APDU OUTPUT: BlockNumber End block size FF 70
                public void GetBaseTemplateInfo(APDU apdu) throws ISOException
                {
                        byte buffer[] = apdu.getBuffer();
                        if( (buffer[ISO7816.OFFSET_P1]!=(byte)00) &&
(buffer[ISO7816.OFFSET_P2]!=(byte)0x7C) )
                                ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
                        buffer[5] = BlockNumber;
                    buffer[6] = EndBlockSize;
                        buffer[7] = (byte)0xFF;
                    buffer[8] = (byte)0x70;
                        apdu.setOutgoingAndSend((short)5,(short)4);
                }

//*********************************Matching Section.****************************************
      // Max. number of minutiae = 25;
      private short main_template[] = new short[100];// i,j,t,f. in EEPROM.4*25 (Memory to store the secured
minutiae template during enrolment in non-volatile memory. The template will never be released to the
outside of the smartcard.)
      private byte base_template[] = new byte[450]; //compressed {d[3],n[3],o[3],f[3],t[3]}15*25 in EEPROM
18*25 compressed form. (Memory to store the open template during enrolment in non-volatile memory.)
      private byte n_template[] = new byte[25*3];
      private byte i_RidgeFreq,j_RidgeFreq;
      private byte NOMI,NOMJ; // number of minutiae.
                private short mbi[]= new short[4*25],mbj[]=new short[4*25];

                //private short n_buffer[];// = new short[150]; // in Transient memory.

                //**********Constants Area.
      private final static byte RidgeFreqTh = 6;
      private final static byte MNMO =3;
      private final static short MAXI = 0x7FE0;
      private final static short MAXF = 0x7FF0;
      private final static short BF  = 2100; //2250 //pi/8 //(float) 0.392699     //PI8 0.523599   //PI6 0.314159
         //PI10 0.261799 //PI12 0.523599 //PI6
      private final static short BO3          = 1800; // pi/10 //(float) 0.261799          //PI12
                private final static short BD3       = 60; //*(Iscale/10); //55*(Iscale/10); // (11/2)*10 //5.5

      //****************** "Static" members for user info.*******************
      private byte UserPhoto[] = new byte[4096]; // Reserve 4K for photo.
      private byte UserName[] = new byte[30];
      private byte UserAddress[] = new byte[120];
                private byte UserPassword[] = new byte[20];
      private byte UserID[] = new byte[15];
                private byte UserSex = (byte)0; //0: undefine,1: Male, 2: Female,
                                                                        //3: Tran-sexual-male,4:
Tran-sexual-female,
                                                                        //5-255: Don't know?
      private byte UserInfoLock = (byte)0; //1-lock, 0-Unlock. NB. Once it locks, there is no way to unlock it.
      private byte UserValid = (byte)0;    // 0 - invalid, 1 Valid.
      //****************** Final Matching variable.*******************
      private short mi_xi=(byte)0,mi_xj=(byte)0,mj_xi=(byte)0,mj_xj=(byte)0,O_angle=(byte)0
                                ,mi_index=(byte)0,mj_index=(byte)0;
                private short Mindex;
                private byte UserVerifyDone = (byte)0;
                private byte MatchScore = (byte)0x00;

                //****************** Card mode control *******************
                private byte FIDmode = -1;
```

```
        //****************** Base template info.*******************
        private byte BlockNumber=0;
        private byte EndBlockSize=0;

// Lock User's Info. Once it locks, there is no way to unlock it.
//          CLA INS P1 P2 LA (BYTES)       .....      LE(return bytes)
//APDU INPUT: CLA INS FA 25 02                              .....      --
//APDU OUTPUT: FF 70 (lock User's info. successfully.
//Exception: ISO7816.SW_INCORRECT_P1P2(invalid unlock pin)
//Exception: ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED(User's info. has been locked.)

        private void lockUserInfo(APDU apdu) throws ISOException
        {
                byte buffer[] = apdu.getBuffer();
                if((buffer[ISO7816.OFFSET_P1]!=0xFA)||(buffer[ISO7816.OFFSET_P1]!=0x25))
                        ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
                if(UserInfoLock == 1)

ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                else
                        UserInfoLock = 1;
                buffer[5] = (byte)0xFF;
                buffer[6] = (byte)0x70;
                apdu.setOutgoingAndSend((short)5,(short)2);
        }
// SetUserInfo(..)
//          CLA INS P1 P2     LA (BYTES)       .....      LE(return bytes)
//APDU INPUT: CLA INS 00 type          02                          .....      --
//APDU OUTPUT: FF 70 (Set User's info. successfully.
//Type :            1-User Name,2-address,3-UserID,4-password,5-sex
//Error : P2 < 5 ISO7816.SW_WRONG_P1P2 exception.
//Note: Should finish matching first and the user has been verified by the matching engine.
//Otherwise, SW_SECURITY_STATUS_NOT_SATISFIED exception will be returned.
        private void SetUserInfo(APDU apdu) throws ISOException
        {
                byte buffer[] = apdu.getBuffer();
                byte array[],i,length;
                byte P2 = buffer[ ISO7816.OFFSET_P2];
                if(P2>5 || P2<0)
                        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
                if( P2 ==5)
                {
                        UserSex = buffer[ ISO7816.OFFSET_P1];
                }
                else
                {
                        array = getUserInfoPointer(P2);
                        length = buffer[ISO7816.OFFSET_LC];
                        if( length > array.length )
                                ISOException.throwIt(ISO7816.SW_FILE_FULL);
                        for(i=0;i<length;i++)
                                array[i]=buffer[i+5];
                        if( (array[(byte)(length-2)]!=0xFF) && (array[(byte)(length-1)]!=0x70) )
                          ISOException.throwIt(ISO7816.SW_WRONG_DATA);
                }
                buffer[5] = (byte)0xFF;
                buffer[6] = (byte)0x70;
                apdu.setOutgoingAndSend((short)5,(short)2);
        }
```

```
// GetUserInfo(...)
//          CLA INS P1 P2    LA (BYTES)       .....      LE(return bytes)
//APDU INPUT: CLA INS 00 type          02                          .....      --
//APDU OUTPUT: UserInfo[] + FF 70 (Set User's info. successfully.
//Type :            1-User Name,2-address,3-UserID,4-password,5-sex
//Error : P2 < 5 ISO7816.SW_WRONG_P1P2 exception.
//Note: Should finish matching first and the user has been verified by the matching engine.
//Otherwise, SW_SECURITY_STATUS_NOT_SATISFIED exception will be returned.
        private void GetUserInfo(APDU apdu) throws ISOException
        {
                byte buffer[] = apdu.getBuffer();
                byte array[],i,length,temp;
                if(UserVerifyDone==0 || UserValid==0)

ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
                byte P2 = buffer[ ISO7816.OFFSET_P2];
                if(P2>5 || P2<0)
                        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
                if( P2==5)
                {
                        buffer[5] = UserSex;
                        buffer[6] = (byte)0xFF;
                        buffer[7] = (byte)0x70;
                        length =3;
                }
                else
                {
                        array = getUserInfoPointer(P2);
                        length = buffer[ISO7816.OFFSET_LC];
                        if( length > array.length )
                                ISOException.throwIt(ISO7816.SW_FILE_FULL);
                        for(i=0;i<length;i++)
                        {
                                temp = array[i];
                                buffer[i+5] = array[i];
                                if(temp == (byte)0x70)
                                  break;
                        }
                                length = i;
                }
                apdu.setOutgoingAndSend((short)5,(short)length);
        }


        private byte[] getUserInfoPointer(byte type)
        {
           switch(type)
                {
                case 1:
                        return UserName;
                case 2:
                        return UserAddress;
                case 3:
                        return UserID;
                case 4:
                        return UserPassword;
                default:
                        return null;
                }
                //return null;
```

```
                }

                // Function for testing only, MUST delete for release version.
                private void UnlockUserInfo(APDU apdu)
                {
                                UserInfoLock = 0;
                }
```

(Function to save the secured minutiae template into the smartcard)

/* The receive_main_template(…) function is used to receive the secured portion of the minutiae template from the PC terminal during enrolment. The received template will be stored in the secured memory and will never be released to the outside of the smartcard. During verification, this template will be transformed to aligned enrolment template first, and then the aligned template will be used to compare against the aligned query template to find out the similarity (matching score) inside the smartcard. All operations related to this secured portion of minutiae template shall only be executed inside the smartcard. In this source code, the secured portion of the minutiae template is named as main_template. */

```
        //receive_main_template(byte[]buffer);
        //          CLA INS P1 P2 LA (BYTES)        .....      LE(return bytes)
        //APDU INPUT: CLA INS FC 32 (size)     ...main_template..  02
        //APDU OUTPUT: FF 70
    public void receive_main_template(APDU apdu)//byte[]buffer)// can be apdu[] buffer in smart card.
    {
                byte buffer[] = apdu.getBuffer();
                short Nbyte;
                Nbyte = apdu.setIncomingAndReceive();
      DecompressionMain(buffer,NOMI);
                buffer[5]=(byte)0xFF;
                buffer[6]=(byte)0x70;
                apdu.setOutgoingAndSend((short)5,(short)2);
    }
/*
    public void SetLocalTemplateInfo(byte nm,byte rf)
    {
        NOMI = nm;
        i_RidgeFreq = rf;
    }
*/
        //GetAuxilaryMinutiae(APDU apdu);
```

/*This function is used to get the index of particular minutia in enrolment template which has been identified for alignment purpose in the alignment process in the PC terminal. This function records down the index and find the corresponding coordinate of the minutia in the enrolment template (inside the secured portion of minutiae template) for later on-card alignment process during matching. This function will not release any information related to secured template to the outside world. */

```
 //Input format: |Index A 8-bit|
   //Output format: |Ai H8|Ai L8|Aj H8|Aj L8|
        //          CLA INS P1 P2 LA (BYTES)        .....      LE(return bytes)
        //APDU INPUT: CLA INS  A 00   00                              .....      04
        //APDU OUTPUT: apdu[0--4]
        public void GetAuxilaryMinutiae(APDU apdu)
   {
                byte buffer[] = apdu.getBuffer();
        short id_A;
        short m_xi,m_xj;
        id_A = (short)(buffer[ISO7816.OFFSET_P1]);
        m_xi = main_template[id_A];
        m_xj = main_template[(short)(id_A+1)];
        buffer[5] = (byte)((m_xi>>8)&0x00FF);
```

```
        buffer[6] = (byte)((m_xi&0x00FF));
        buffer[7] = (byte)((m_xj>>8)&0x00FF);
        buffer[8] = (byte)((m_xj&0x00FF));
                    apdu.setOutgoingAndSend((short)05,(short)04);
    }

        /* The Function SendPartialMain(...) is used to get certain information of minutiae as auxiliary
        information for alignment. This function is an optional function which is not used in later
        version. */
    //format: |lower i(8-bit)| lower j(8-bit)| higher i(4-bit) + lower j(4-bit)|
    //       + |Higher f(8-bit)| lower f(8-bit)|
        //SendPartialMain(APDU apdu)
        //          CLA INS P1   P2   LA (BYTES)       .....              LE(return bytes)
        //APDU INPUT: CLA INS N1   00   (1-8)       cord(i)[](1-8)..     5*N1
        //APDU OUTPUT: apdu[14--54]
        // N1 : number of id(i,j).
      public void SendPartialMain(APDU apdu) throws ISOException //byte cord[],byte packedarray[],byte
num_core)  //,short mi_verify[])
      //packedarray can be apdu buffer.
      {
          byte buffer[] = apdu.getBuffer();
          short index=0,count=0;
          short i,j;
          byte k,num_core = buffer[ISO7816.OFFSET_P1];

          if(num_core>8)
                                    ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
          //{
          //   System.out.println("SendPartialMain(...): Invalid no. of minutiae. length = "+length);
          //   return;// error
          //}
//          try
//          {
          for(k=0;k<num_core;k++)
          {
             index = (short)(buffer[k+5]*4);
             i = main_template[index];
             j = main_template[(short)(index+1)];
             buffer[(short)(14+(count++))] = (byte)(i&0x00FF);
             buffer[(short)(14+(count++))] = (byte)(j&0x00FF);
             buffer[(short)(14+(count++))] = (byte)(((i&0x0F00)>>4)+(((j&0x0F00)>>8)&0x000F));
             i = main_template[(short)(index+3)];
             buffer[(short)((short)14+(short)(count++))] = (byte)((i&(short)0xFF00)>>(byte)8);
             buffer[(short)(14+(count++))] = (byte)(i&0x00FF);
          }
//          }catch(ArrayIndexOutOfBoundsException e)
//          {
//             System.out.println("Array index out of bound: index = " + index+", count = " + count);
//          }
          buffer[(short)(14+(count++))] = (byte)0x55;
          buffer[(short)(14+(count++))] = (byte)0xAA;//Stop bytes (55 AA)
                              apdu.setOutgoingAndSend((short)14,(short)count);
      }


      // 4 packets involved. rewrite to packet mode to transmit data.
      // or use commit buffer method.
      public void construct_n_base_template()//short baselength,byte buffer[])
      {
        short i,index,index1,PackedDataH,PackedDataL;
```

```
        byte j;
        //short length = (short)buffer.length;
//          for(i=0;i<baselength;i++)
//          {
//             base_template[i]=buffer[i];
//          }
        // Extract the number of ridge between two minutiae for matching.
        for(i=0;i<NOMI;i++)
        {
         index = (short)(i*3);
         index1 = (short)((i*18)+8);
         PackedDataH =0;
         PackedDataL=0;
         for(j=0;j<4;j++)
         {
           if(j<2)
            PackedDataL += (int)(((int)base_template[(short)(index1 + j)]&(int)0x00ff)<<(short)(j<<3));
           else
            PackedDataH += (int)(((int)base_template[(short)(index1 + j)]&(int)0x00ff)<<(byte)((j-2)<<3));
         }
         n_template[index++] = (byte)((PackedDataH>>2) & 0x00FF);
         n_template[index++] = (byte)(((PackedDataH&0x01)<<7) + ((PackedDataL >>1)&0x007F));
         n_template[index] = (byte)(PackedDataL & 0x00FF);

        }
      }
```

// LoadMatchingVariables(APDU apdu) **(Function to load the aligned template from the PC).**

/* The *LoadMatchingVariablesC* (…) function is used to load the aligned query template from the PC terminal after the alignment process is completed on the PC side. The template will be stored in volatile memory in the smartcard. This template will be used to calculate the matching score during the matching process.
***/**

```
                // CLA INS P1 P2 LA (BYTES)      ....      LE(return bytes)
                // APDU INPUT: CLA INS SM EM Size     Matching Variables      --
                // APDU OUTPUT: --
                //  SM: Start Minutiae. min. value = 0;
                //  EM: End Minutiae.   min. value > SM;
                // Matching Variable using the format:
                //          |t(4-bit)+d(12-bit)|o(16-bit)H,L|f(16-bit)H,L|
                // Using Mindex store the index of last updated mbj[..]
                // Currently using EEPROM to store mbj[...], we can use ram to store in order
                // to increase the speed.
                // if P1>NOMJ | P2>NOMJ | P1>P2 , it returns ISOexception: SW_INCORRECT_P1P2
    public void LoadMatchingVariables(APDU apdu) throws ISOException  //byte nj,byte buffer[]) //,short
mbj_temp[]) // apdu buffer again!
    {
                        byte buffer[] = apdu.getBuffer();
                        byte SM = buffer[ISO7816.OFFSET_P1];
                        byte EM = buffer[ISO7816.OFFSET_P2];
                        if( (SM>NOMJ) || (EM>NOMJ) || (EM>SM))
                                ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
                        short Nbyte;
                        short Nbytes = apdu.setIncomingAndReceive();
                        // Note: timeout setting may need to increase.
        short i;
        short count=5,temp,index;
        //NOMJ = nj;
```

```
        for(i=SM;i<EM;i++)
        {
          index = (short)(i<<2);
          temp = (short)(buffer[count++]<<8);
          temp += ((short)buffer[count++]&0x00FF);
          mbj[index] = (short)(temp&0x0FFF);    //d,t
          mbj[(short)(index+3)]= (short)((temp>>12)&0x000F);
          temp = (short)(buffer[count++]&0x00FF);
          temp = (short)(temp<<8);
          temp += (short)((short)(buffer[count++])&0x00FF);
          mbj[(short)(index+1)] = temp;
          temp = (short)((short)buffer[count++]<<8); //o
          temp += (buffer[count++]&0x00FF);
          mbj[(short)(index+2)] = temp;  //f
        }
    }
                //Since APDU byffer has an offsets always equals to 5.
    private void GetBaseInfo(byte nj_template[])
    {
       short temp,index;
        index = (short)((NOMJ<<(byte)1)+NOMJ+(byte)5);// 5 if the offset of the APDU buffer.
        temp = (short)(((short)(nj_template[index++])<<(byte)8)&(short)0xff00);
        temp += (short)((short)nj_template[index++]&0x00ff);
        mi_xi = temp;
        temp = (short)(((short)(nj_template[index++])<<(byte)8)&(short)0xff00);
        temp += (short)((short)nj_template[index++]&(short)0x00ff);
        mi_xj = temp;
        temp = (short)(((short)(nj_template[index++])<<(byte)8)&(short)0xff00);
        temp += (short)((short)nj_template[index++]&0x00ff);
        mj_xi = temp;
        temp = (short)(((short)(nj_template[index++])<<(byte)8)&(short)0xff00);
        temp += (short)((short)nj_template[index++]&(short)0x00ff);
        mj_xj = temp;
        temp = (short)(((short)(nj_template[index++])<<(byte)8)&(short)0xff00);
        temp += (short)((short)nj_template[index++]&(short)0x00ff);
        O_angle = temp;
        mi_index = (short)(nj_template[index++]);
        mj_index = (short)(nj_template[index++]);
    }


        // matching function.************************************
        (Function to perform final on-card fingerprint matching using aligned minutiae template)
        /* The Match (...) function is used to match the aligned query template which has be received
        by using LoadMatchingVariables(...) function. Before the computation of matching, the on-
        card alignment procedure for the secured portion of enrolment template will be executed
        inside the smartcard first. Once the alignment of the enrolment template is completed, the
        smartcard will collate the aligned enrolment template with the aligned query template in order
        to find out the overall similarity as matching score. The matching score will be compared to
        the internal security threshold to determine the query is from genuine user or imposter to
        grant or deny the subsequent transaction respectively.
        */
        //          CLA INS P1 P2 LA BYTES        .....      LE(return bytes)
        //APDU INPUT: CLA INS 00 00 (size) ...nj_template... 03
        //APDU OUTPUT: Matching Score (FF 70) valid return code.
        public void match(APDU apdu) throws ISOException //short rfj,byte nj_template[]) // nj_template:
apdu buffer.
        {
        byte nj_template[] = apdu.getBuffer();
        short MatchingScore=0;
```

```
        byte TestBase;
        short nm;
        short Nbytes;
        Nbytes = apdu.setIncomingAndReceive();
        if(Nbytes != (short)((NOMJ<<1)+NOMJ))
                ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
  //j_RidgeFreq = rfj;
        if (labs((short)(i_RidgeFreq-j_RidgeFreq))>RidgeFreqTh)
                ISOException.throwIt(ISO7816.SW_DATA_INVALID);
        if (NOMJ < MNMO) ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
        if (NOMI>NOMJ)
                nm = NOMI;
        else
                nm = NOMJ;
     GetBaseInfo(nj_template);
        lalign(); // This lalign(..) function performs on-card alignment of the secured portion of minutiae
(main_template) using the minutiae index received by GetAuxilaryMinutiae (...) function which
executed earlier.
        MatchingScore = lmatchm(nj_template); //ni,nj; // Matching in polar coordinate.
        if (MatchingScore>200) MatchingScore = (short) ((short)(MatchingScore-200)/(short)(nm-2));
          else MatchingScore = 0;
        if(MatchingScore>100)
                MatchingScore=100;
        MatchScore = (byte)MatchingScore; //store matching score in card.
        UserVerifyDone =1; // Finish Verification
        if(MatchingScore>20)
          UserValid=1;
        nj_template[5] = (byte)MatchingScore;
        nj_template[6] = (byte)0xFF;
        nj_template[7] = (byte)0x70;
        apdu.setOutgoingAndSend((short)5,(short)3);
         //return the MatchingScore;
        }

/*********************************************************************
                Aligning minutiae
                AUTHOR:         JIANG XUDONG
*********************************************************************/
        private void lalign()
        {
                byte     i;
                short SA1,SA2,SA3,SA4,SA5,SA6,SA7;
                short MMindex,s_index; //b_index,
                short alpha,beta;
                SA1 = mi_xi;
                SA2 = mi_xj;
                SA3 = mj_xi;
                SA4 = mj_xj;
                SA7 = (short)(O_angle);
                for (i=0; i<NOMI; i++)
                {
                        if(i==mi_index)
                                continue;
                        MMindex = (short)(i*4);
                        SA5 = (short)(main_template[MMindex] - SA1);//ii
                        SA6 = (short)(main_template[(short)(MMindex+1)] - SA2);//ij
                        s_index = (short)(i*4);
                        mbi[s_index] = (short) Dist_calcA(SA5,SA6);       //d
                        mbi[(short)(s_index+1)] = atan2(SA5,SA6);         //o
                        mbi[(short)(s_index+2)] = main_template[(short)(MMindex+3)];//f
```

```
                        mbi[(short)(s_index+3)] = main_template[(short)(MMindex+2)]; //t
                }
        }


/*****************************************************************
                local Matching
                AUTHOR:         JIANG XUDONG
*****************************************************************/
// set nj_minutiae[] template offset to 5.
private short lmatchl(byte i,byte j,byte nj_minutiae[])
{
        //short             n11, n22, n33, n12, n21, n23, n32;
        short       SA1, SA2, SA3, SA4, SA5, SA6, SA7;
        short     fd[] =new short[9];
        short     mfd,s_index,MMindex;
        MMindex = (short)((i<<1)+i);
        s_index = (short)((j<<1)+j+5); // 5 tes for apdu offset. //9-0,10-1,11-2
        SA1 = labs((short)(nj_minutiae[s_index]  - n_template[MMindex]));
        SA2 = labs((short)(nj_minutiae[(short)(s_index+1)] - n_template[(short)(MMindex+1)]));
        SA3 = labs((short)(nj_minutiae[(short)(s_index+2)] - n_template[(short)(MMindex+2)]));
        SA4 = labs((short)(nj_minutiae[s_index]  - n_template[(short)(MMindex+1)]));
        SA5 = labs((short)(nj_minutiae[(short)(s_index+1)] - n_template[MMindex]));
        SA6 = labs((short)(nj_minutiae[(short)(s_index+1)] - n_template[(short)(MMindex+2)]));
        SA7 = labs((short)(nj_minutiae[(short)(s_index+2)] - n_template[(short)(MMindex+1)]));
        fd[0] = (short)(SA1+SA2);//(n11+n22);
        fd[1] = (short)(SA1+SA6);//(n11+n23);
        fd[2] = (short)(SA4+SA6);//(n12+n23);
        fd[3] = (short)(SA1+SA7);//(n11+n32);
        fd[4] = (short)(SA1+SA3);//(n11+n33);
        fd[5] = (short)(SA4+SA3);//(n12+n33);
        fd[6] = (short)(SA5+SA7);//(n21+n32);
        fd[7] = (short)(SA5+SA3);//(n21+n33);
        fd[8] = (short)(SA2+SA3);//(n22+n33);

        mfd = 1024;
        //if (fd<mfd) mfd = fd;
        if (fd[0]<mfd) mfd = fd[0];
        if (fd[1]<mfd) mfd = fd[1];
        if (fd[2]<mfd) mfd = fd[2];
        if (fd[3]<mfd) mfd = fd[3];
        if (fd[4]<mfd) mfd = fd[4];
        if (fd[5]<mfd) mfd = fd[5];
        if (fd[6]<mfd) mfd = fd[6];
        if (fd[7]<mfd) mfd = fd[7];
        if (fd[8]<mfd) mfd = fd[8];
   if (mfd<2) return(20);
        else if (mfd<3) return(10);
        else if (mfd<4) return(5);
        else return(0);

}
/*****************************************************************
                                        Match the minutiae
                                        AUTHOR:         JIANG XUDONG
*****************************************************************/


private short lmatchm(byte nj_minutiae[])//nj_minutiae will be remapped to apdu buffer.
{
```

```
        byte i, j;
        short s_index,t_index;
        short ti;
        short k,kk=0;
        short dds,di,dj;
        short   dfs, dos, bot, bdt,oi, fi; //dos--angle.
      bot = 1800; //B03 = 1800;//BF=2100;
      bdt = 12;//BD3= 8;
      for (i=0; i<NOMI; i++) {
            if(i==mi_index)
                  continue;
            s_index = (short)(4*i);
            di = mbi[s_index];
            oi = mbi[(short)(s_index+1)];
            fi = mbi[(short)(s_index+2)];
            ti = mbi[(short)(s_index+3)];
            k = 0;
            for (j=0; j<NOMJ; j++) {
                  if(j==mj_index)
                        continue;
                  t_index = (short)(j*4);
                  dj = mbj[t_index];
                  if(dj<0)
                        continue;
                  dds = labs((short)(dj - di));
                  if (dds<bdt)
                  {
                dos = labs((short)( mbj[(short)(t_index+1)] - oi ));
                        if (dos>18000) dos = (short)(18000 + (18000- dos));
                        if (dos<bot)
                        {
                              dfs =  labs((short)(mbj[(short)(t_index+2)] - fi ));
                              if (dfs>18000) dfs = (short)(18000+ (18000 - dfs));
                              if (dfs<BF)
                              {
                                    k = 60;
                                    if (mbj[(short)(t_index+3)] == ti) k += 20;
                                    k += lmatchl(i,j,nj_minutiae);
                                    kk += k;
            mbj[t_index]=-1;
                                    if(k>0) break;
                              }//endof if(dfs<BF)
                         }//endof if(dos<bot)
                   }//endof if(dds<bdt)
            }//for(j..)
      }//for(i..)
      return kk;
}
            //Since Zmain is apdu buffer, offset to index =5;
    private void DecompressionMain(byte Zmain[],byte N_Minutiae)
    {
      byte i;
      short index=5,index1=0,temp,temp1;
      for(i=0;i<N_Minutiae;i++)
      {
         temp = (short)((((short)Zmain[index])&0x00FF)<<8);
         temp = (short)(temp + (((short)(Zmain[(short)(index+1)]))&0x00FF));
         main_template[(short)(index1)] = (short)((temp >> (short)7)&0x01FF); //i
         temp1 = (short)((temp & (short)0x7F)<<2); //j
         temp = Zmain[(short)(index+2)];
```

```
    main_template[(short)(index1+1)] = (short)(temp1 +(short)((temp & (short)0xC0)>>6));
    main_template[(short)(index1+2)] = (short)(temp & 0x0F);            //t
    temp1 = (short)((((short)Zmain[(short)(index+3)])&0x00FF)<<8);      //f
    main_template[(short)(index1+3)] = (short)(temp1 + (short)((Zmain[(short)(index+4)])&0x00FF));
    index += 5;
    index1 += 4;
  }
}


    private final static short max_value= 32760;
    // function : atan(x)
    // x - integer(must be multiplied by 1000)
    // return value : atan(x) degree (+/- 0.5 degree).
    //              format: 000.00
    // range of x : +/- (0 to max_inf.)
    // resoution of input: 0.01
    // range of return value : -90.00 to 90.00
    // private short temp,x1,y1,z1,a1,delta;

    private short atan(short x)
    {
            short temp,delta,x1;
            short value = -30000;
            temp = labs(x);
            if(temp > 31000)
                    return 9000;
            delta = (short)(temp-((temp/(short)(10))*(short)(10)));
            if(temp>1000)       //    x>1;
            {
               temp = (short)(32760/temp);
                temp = (short)(temp * 31);
            }
       else
            {
                    x1 = (short)(temp/10);
                    if(x1==100)
                            value =(short)(sgn(x)*4500);
                    else
                    {
                    temp = atan_value[x1];
                    temp = (short)(temp + (short)((atan_value[(short)(x1+1)]-temp)*delta)/(short)(10));
                    value = (short)(sgn(x)*temp);
                    }
            }
            if(temp >1000)
                     temp =1000;
            if(value == -30000)
            {
                    delta = Delta(temp);
                    x1 = (short)(temp/10);
                    if(x1==100)
                      value= (short)(sgn(x)*4500);
                    else
                    {
                    temp = atan_value[x1];
        temp = (short)(temp + (short)((atan_value[(short)(x1+1)]-temp)*delta)/(short)(10));
                    value = (short)(sgn(x)*(9000 - temp));
                    }
                    //value = (short)(sgn(x)*(9000 - atan_value[temp]));
```

```
        }
        //return (short)(Math.atan((float)(x)/1000.0f)/3.141592654f*18000.0f);
        return value;
}



// function : atan2(y,x)
// x,y - integer
// return value : atan2(y/x) degree (+/- 0.5 degree).
// range of x : +/- (0 to max_inf.)
// resoution of input: 1
// Return range : -180.00 to 180.00

private short atan2(short y,short x)
{
        short x1,y1,temp;
          x1=labs(x);
          y1=labs(y);
          if(x==0 || y==0)
                  return 0;
          if(x==0 && y>0)
                  return 90;
          if(x==0 && y<0)
                  return -90;

          if((x1>y1 && x1>15000)||(y1 > max_value))
                  temp = (short)(((short)y1/(short)(((short)x1/(short)100))*(short)10));
          else
          {
                  if((short)(y1/x1)>55)
                          temp = 31500;
                  else
                          temp = (short)(((short)(y1*100)/x1)*10);
          }
    temp =  atan(temp);

          if(x>0 && y>0)
          {
                  if(temp<0)
                  temp = (short)(-temp);
          }
          else
          if(x<0 && y>0)
                  temp = (short)(18000 - temp);
          else
          if(x<0 && y<0)
          {
                  if(temp>0)
                  temp =  (short)(-18000 + temp);
                  else
                  temp =  (short)(-18000 - temp);
          }
          else
          if(x>0 && y<0)
          {
                  if(temp>0)
                          temp = (short)(-temp);
          }
```

```
                if(temp>(short)(18000))
                        temp = (short)(18000 + (18000 - temp));
                if(temp<(short)(-18000))
                        temp = (short)(18000 - (18000 + temp));
                // Checking the precission of the Atan to avoid any truncation error.

              // short test = (short)((Math.atan2(y,x)*180.0f / 3.141592654f)*100);
              //if(Math.abs(test - temp)>500)
              //{
              //        System.out.println("Atan Precission Error");
              //        System.out.println("x="+x+",y="+y+",atan2 = "+temp+",real atan2="+ test);
              // }
              //if(sgn(test)!=sgn(temp))
              //{
              //        System.out.println("Atan Sign Error");
              //        System.out.println("x="+x+",y="+y+",atan2 = "+temp+",real atan2="+ test);
              // }

                //return test;
                return temp;
        }

        private short short_div(short a,short b)
        {
      short c,d,e,f;
              //if(true)
              c = (short)(a/b);
              d = (short)(a%b);
              c = (short)(c*(short)1000);
              e = (short)(  (short)(d* (short)1000) / b);
              f = (short)(  (short)(d* (short)1000) % b);
//            e = (short)((d *(short)((short)1000)/b))
//            f = (short)((d *(short)((short)1000)%b));
              if(f!=(short)0)
              {
                if((short)(b/f)>=(short)2)
                        d=(short)(d+(short)1);
              }
              c = (short)(c+e);
              if(c<0)
                        c=(short)(-c);
              return c;
        }
        private short short_div1(short a,short b)
        {
      short c,d,e,f;
              //if(true)
              c = (short)(a/b);
              d = (short)(a%b);
              c = (short)(c*(short)1000);
              e = (short)(  (short)(d* (short)1000) / b);
              f = (short)(  (short)(d* (short)1000) % b);
//            e = (short)((d *(short)1000)/b);
//            f = (short)((d *(short)1000)%b);
              if(f!=0)
              {
                if((short)(b/f)>=(short)2)
                        d=(short)(d+(short)1);
              }
              c = (short)(c+e);
```

```
        return c;
}

private short Delta(short a)
{
        return (short)(a - (a/(short)10)*(short)10);
}


private short Dist_calcA(short xa,short ya) throws ISOException
{
        short a_temp,temp1,temp;
        short x1,y1,delta,xa1,ya1;
        short t1;
        if(xa==(short)0)
                return ya;
        if(ya==(short)0)
                return xa;
        xa1 = labs(xa);
        ya1 = labs(ya);

        if(ya1>xa1)
        {
                x1=ya1;
                y1=xa1;
        }
        else
        {
                x1=xa1;
                y1=ya1;
        }

        if((short)(x1/y1)>(short)6)
           return x1;
        else
           temp = short_div(y1,x1);
        if(temp<(short)0)
                temp = (short)(-temp);

        if(temp<=(short)1000 && temp>=(short)0)
        {
                delta = Delta(temp);
                t1= (short)(temp/(short)10);
                if(t1==100)
                temp = (short)(4500);
                else
                {
                temp = atan_value[t1];
                a_temp = atan_value[(short)(t1+(short)(1))];
                temp = (short)(temp + (short)((short)(a_temp-temp)*delta)/(short)(10));
                }
           a_temp = (short)(temp/(short)10);
           if((short)(temp - (a_temp)*(short)10)>(short)4)
                  temp = (short)(a_temp+1);
           else
                  temp = a_temp;

                if(temp > (short)450)
                {
                        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
```

```
                                //System.out.println("Value greater than 45 drgrees:"+temp);
                                //System.exit(1);
                        }
                        delta = Delta(temp);
                        t1 =(short)( temp/(short)10);
                        if(t1==(short)45)
                        temp = (short)(707);
                        else
                        {
                        temp = sin_value[t1];
                        temp = (short)(temp + (short)((short)(sin_value[(short)(t1+(short)1)]-
temp)*delta)/(short)(10));
                        }
                }

                else
                        temp = (short)1000;
                if(temp == (short)0)
                        return (short)(y1);

                temp = short_div(y1,temp);
                //short test = (short)(Math.sqrt(xa1*xa1 + ya1*ya1));
                //if((test - temp)>6)
                //      System.out.println("Precission Error:x="+xa1+",y="+ya1+",result="+temp+",Real
Result="+test);
                //return test;
           if(temp<(short)0) return (short)(-temp);
                return temp;
        }
        private short labs(short a)
        {
                if(a<(short)0)
                        return (short)(-a);
                return a;
        }

        private short sgn(short x)
        {
                if(x<(short)0)
                        return (short)-1;
                else
                        return (short)1;
             //return (x<0)? -1:1;
        }


}// End of Oppurse
```